|  |  |  |  |
| --- | --- | --- | --- |
|  | | logo_black.gif | |
|  | 200925978 |  | 1664 |
|  | GEOG5990M  Programming for GIS: Core Skills |  |  |
|  | Assessment 2 |
|  | Andy Turner |

|  |
| --- |
|  |
| 1. |
| 2. |
| 3. |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

**Intention of software:**

An ever increasing number of people and organisations use and engage with social media sites such as Twitter (Ahmed et al, 2017) providing a vast amount of data with the potential to be used within research. The nature of social networking sites, onto which people tend to post about their thoughts, feelings or opinions (Ahmed et al, 2017), stands this type of data apart from more traditional forms of data as it can provide real time insights into societal attitudes.

This software has therefore been developed with the intention of creating a means of accessing and analysing twitter data. In particular, to use sentiment analysis to gage general views towards certain topics.

**Development process:**

Resolution of issues during the development of this code has been aided by the use of ‘Stackflow’ and the documentation of the modules/packages used within the code.

Modules/Packages

Having found ‘tweepy’ to be a suitable and popular package for the extraction of twitter data I first had to install the latest version using ‘pip install tweepy == 3.7.0’ within the terminal. However, having done this, I had issues importing ‘tweepy’ into Spyder as no module named ‘tweepy’ could be found which suggested that the install had not been successful. However, I found that the issues actually stemmed from having multiple versions of python installed on my Mac, and on deleting and reinstalling the necessary programmes I was able to successfully install and import ‘tweepy’.

API Access & Authentication

Having tried to work with Twitter data using other software in the past I already had access to a Twitter API so was able to set up a connection using the necessary keys/tokens quickly. Originally this authentication process was in the source code but I later decided that, given the private nature of the API details and the importance of them being entered correctly by the user before attempting to run the code, it would be more appropriate for this step to rely on an additional ‘authentication.py’ file.

Acquiring Tweets

I began by testing the Twitter API access with a simple task, accessing my twitter timeline, using the api.home\_timeline() tweepy method. After successfully doing this I then developed the code further to search for tweets by user and by key word and was able to get my head around the JSON format that tweets are returned it, learning how to access and print specific bits of a tweets such as the location or time it was created. At this stage, using various different methods within the ‘api.’ class alone was only retrieving the default 20 tweets at a time and was limited to 200.

I therefore added in the use of tweepy’s cursor method which automatically handles pagination, allowing for any number of tweets, as assigned to the ‘items’ parameter to be iterated through. Rather than simply printing the tweets I was returning as I had been, the next step was to append each tweet to a list before writing to a data frame where each JSON element could be inputed into a designated column. Having the twitter data stored in this format would then allow for it to be analysed and plotted.

Setting search parameters

At first, the desired search criteria (‘q’, the word to search for and ‘lang’, the language to search for) were assigned directly to the parameters:

for tweet in tweepy.Cursor (api.search, q = ‘Brexit’, lang = ‘en’).items(100):

results.append(tweet)

However, to make it clearer for the user to alter the search criteria I created variables to which to assign the criteria, and assigned these variables to the parameters within the API search:

query = ‘Brexit’

language = ‘en’

n\_tweets = 10

for tweet in tweepy.Cursor (api.search, q = query, lang = language).items(n\_tweets):

results.append(tweet)

Plot bar chart/pie chart

In order to plot the outcomes of the sentiment analysis I utilised the Python plotting library, matplotlib as I felt the results would be easier to interpret when visualised.

I initially tried to plot the bar chart by pulling data directly from ‘data\_set’ but realised that a calculation, to determine how many tweets of each sentiment class there were, was needed first. Pandas method value\_count was used to return the count of each unique value in the ‘Sentiment Class’ column of ‘data\_set’ and the result was assigned to a variable that could be passed into the plot functions.

Within the value\_count documentation I found that setting the ‘normalize’ parameter to True would calculate the frequency of each sentiment class rather than count which provided the values necessary to then plot a pie chart.

Plots were cropped slightly at the edges when saved to png which was resolved by setting the ‘bbox\_inches’ parameter to ‘tight’ to ensure whole figure its saved.

Searching by User

In my early exploration of tweepy functions I was searching for tweets created by a specified user and wanted to add a way to carry out a search in this way as well as by key word. However, this would require two different tweepy methods, alternative search criteria and would impact aspects of outputs such as titles.

To resolve this I first added a ‘user’ variable to which the user could assign a twitter username to. I then created a ‘search\_by’ variable that the user could set to ‘Word’ or ‘User’. This allowed me to implement ‘if’ statements based on this variable, to differentiate between the code for searching by key word and by user.

if search\_by == 'Word':

for tweet in tweepy.Cursor (api.search, q = query,lang=language).items(n\_tweets):

results.append(tweet)

else:

for tweet in tweepy.Cursor (api.user\_timeline, id=user,).items(n\_tweets):

results.append(tweet)

To resolve the issue of ouput titles, my first solution was to define two separate functions for each output, each with different labels/titles and use ‘if’ statements to control which of the functions to call depending on the type of search carried out:

def bar\_chart\_word(sentiment\_count):

index = np.arange(len(sentiment\_count))

plt.bar(index, sentiment\_count, color = '#7EC0EE')

plt.xlabel('Sentiment Class', fontsize=10)

plt.ylabel('Number of Tweets', fontsize=10)

plt.xticks(index, sentiment\_count.index, fontsize=10, rotation=30)

plt.title("Sentiment Count of Tweets Containing Search Term " + '"' + query + '"')

plt.savefig('/200925978\_Assess\_2/barchart.png', bbox\_inches = 'tight')

plt.show()

def bar\_chart\_user(sentiment\_count):

index = np.arange(len(sentiment\_count))

plt.bar(index, sentiment\_count, color = '#7EC0EE')

plt.xlabel('Sentiment Class', fontsize=10)

plt.ylabel('Number of Tweets', fontsize=10)

plt.xticks(index, sentiment\_count.index, fontsize=10, rotation=30)

plt.title("Sentiment Count of Tweets by " + '"' + user + '"')

plt.savefig('/200925978\_Assess\_2/barchart.png', bbox\_inches = 'tight')

plt.show()

if search\_by == ‘Word’:

bar\_chart\_word()

else:

bar\_chart\_user()

However, this involved a lot of extra lines of code so I simplified it by adding ‘if’ statements into the original output functions to specify different titles depending on the search type.

Adding a GUI

Though adding variables made it easier for the user to input their search criteria I felt that a Graphical User Interface (GUI) would be a more intuitive way for these details to be entered. To create this I used Python’s standard GUI package, tkinter.

Initially I used the ‘entry’ tkinter widget to allow the user to enter a string of text into input boxes labeled “Search by’, ‘Language’, ‘Search Term/Username’ and ‘No. of Tweets’. This method was appropriate for the user to enter the query and number of tweets however, when setting the search criteria I decided that a dropdown menu would be more suitable to ensure that the user could only select either ‘Word’ or ‘Username’. This involved using tkinters OptionMenu class to create a menu, with the contents taken from a dictionary created containing the two options. As the ‘language’ parameters of the tweepy search requires the chosen language to be input in 639-1 language code format I decided that a dropdown menu for this input may also be more suitable and prevent confusion. I therefore created a dictionary of the most commonly spoken language and their language codes and set up a dropdown menu of these codes within the GUI, setting the default to ‘en’ (English).

In order for the code to run I then had to experiment with how to assign the user inputs from the GUI to the variables assigned to the tweepy search parameters. Each input widget was assigned to a variable such as ‘count\_input’ so I initially used these:

query = search\_input

n\_tweets = count\_input

However this returned a 404 Twitter Error response and on printing the user input I realised that I was simply assigning the tkinter variable to the search criteria variables rather than the contents of the entry field. I instead needed to use the ‘get’ method. This returns the current entry text in string format, which could be assigned directly to the search criteria variables for all but the ’n\_tweets’ variable for which the format had to be converted to integer in order to fit the requirements of the ‘items’ parameter.

Importing CSV for users without Twitter API access

Though this software is designed to be used with access to a Twitter API I was conscious that my own consumers tokens etc should be kept private and so the user would need to have or set up access to a Twitter API themselves. In order to provide an example of what the software can do to someone without API access I wrote the results of a search to a CSV file and then added in a section of code that reads in this CSV to replace the data frame that would be created if the code was run in full. As it stands this involves the user uncommenting this section of code and commenting out the sections that aren’t needed. Although this runs fine it is not ideal as the user could make a mistake and so further development of the software could involve an extra step where the user inputs whether they have access to an API or not and the relevant code runs accordingly. My first attempt at doing his, involving defining each section of the code as a function and using an if statement to differentiate between which functions to call depending on the an ‘API\_Access’ variable was set to TRUE or not, was not successful but has potential if developed further.

References:

Ahmed, W., Bath, P. and Demartini, G. 2017. Chapter 4 Using Twitter as a Data Source: An Overview of Ethical, Legal, and Methodological Challenges. In: Woodfield, K., ed. The Ethics of Online Research. Advances in Research Ethics and Integrity (2). Emerald, pp. 79-107.